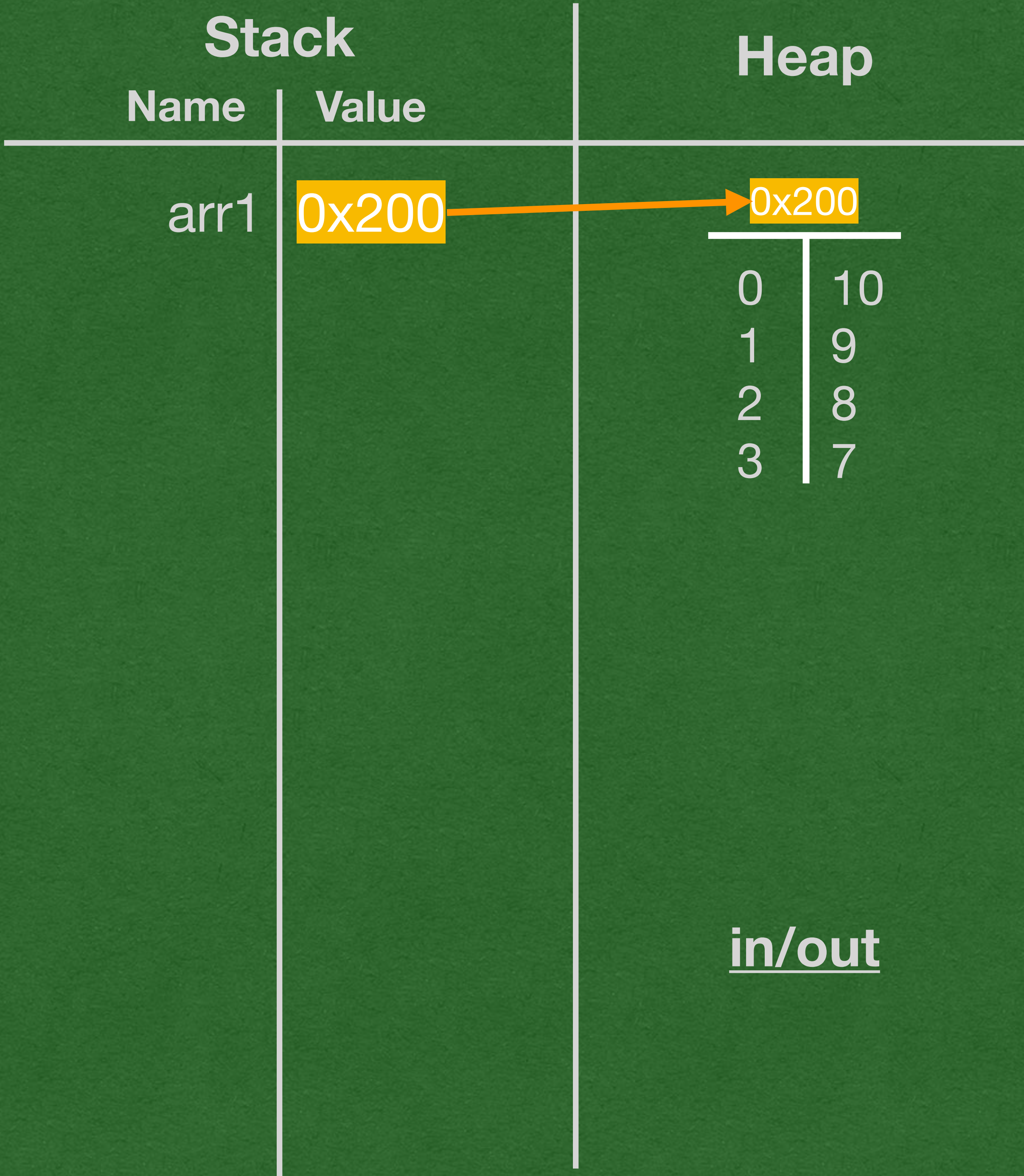


Linked List

ArrayList [And Array]

- Sequential
 - One continuous block of memory
 - Random access based on memory address
 - $\text{address} = \text{first_address} + (\text{element_size} * \text{index})$
- Fixed Size
 - Since memory adjacent to the block may be used
 - Efficient when you know how many elements you'll need to store

- We show an ArrayList on the heap in columns
- Values are all located in one continuous block of memory
- This is actually how ArrayLists [and Arrays] are stored



- This ArrayList stores 32-bit ints (4 bytes) and the ArrayList is stored at memory address 0x200
- Find the element of each value using
- $\text{address} = 0x200 + (4 * \text{index})$
- Easy to find any value, given it's index

Stack		Heap	
Name	Value		
arr1	0x200	0x200	
		0x200	10
		0x204	9
		0x208	8
		0x212	7

in/out

- This is called random access
- Memory is like a giant array
- We call it RAM (Random Access Memory)

Stack		Heap		
Name	Value			
arr1	0x200	0x200		
		0x200	0	10
		0x204	1	9
		0x208	2	8
		0x212	3	7
		<u>in/out</u>		

Linked List

- Sequential
 - Spread across memory
 - Each element knows the memory address of the next element
 - Follow the addresses to find each element
- Variable Size
 - Store new element anywhere in memory

Linked List

- Each value in a list is stored in a separate object on the heap
- Also stores a reference to the next element
- A reference to the list is only a reference to the first value
- Last link stores null
 - We say the list is "null terminated"
 - When we read a value of null we know we reached the end of the list

Linked List

```
package week4;

public class LinkedListNodeInt {
    private int value;
    private LinkedListNodeInt next;

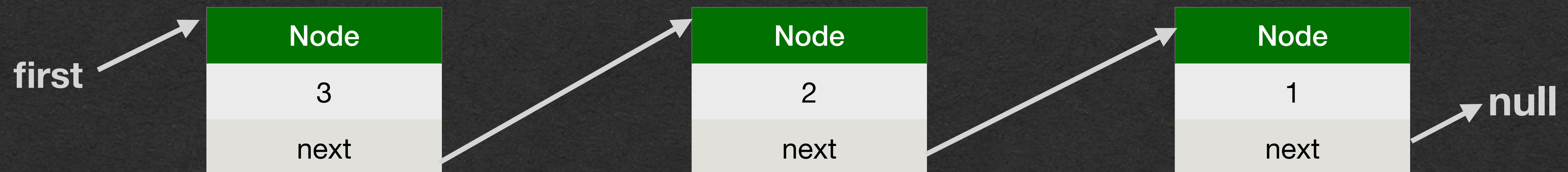
    public LinkedListNodeInt(int value, LinkedListNodeInt next) {
        this.value = value;
        this.next = next;
    }

    public static void main(String[] args) {
        LinkedListNodeInt first = new LinkedListNodeInt(1, null);
        first = new LinkedListNodeInt(2, first);
        first = new LinkedListNodeInt(3, first);
    }
}
```

- We create our own linked list node class
- A node represents one "link" in the list
- The list itself is a reference to the first/head node

Structure

- Each node stores one value of the list
- Each node refers to the next node
- A variable "storing" a list stores a reference to the first node of the list



Memory Diagram

- LinkedListNodeInt -> LLNode
- To save space on the slide

```
public class LLNode {
    private int value;
    private LLNode next;

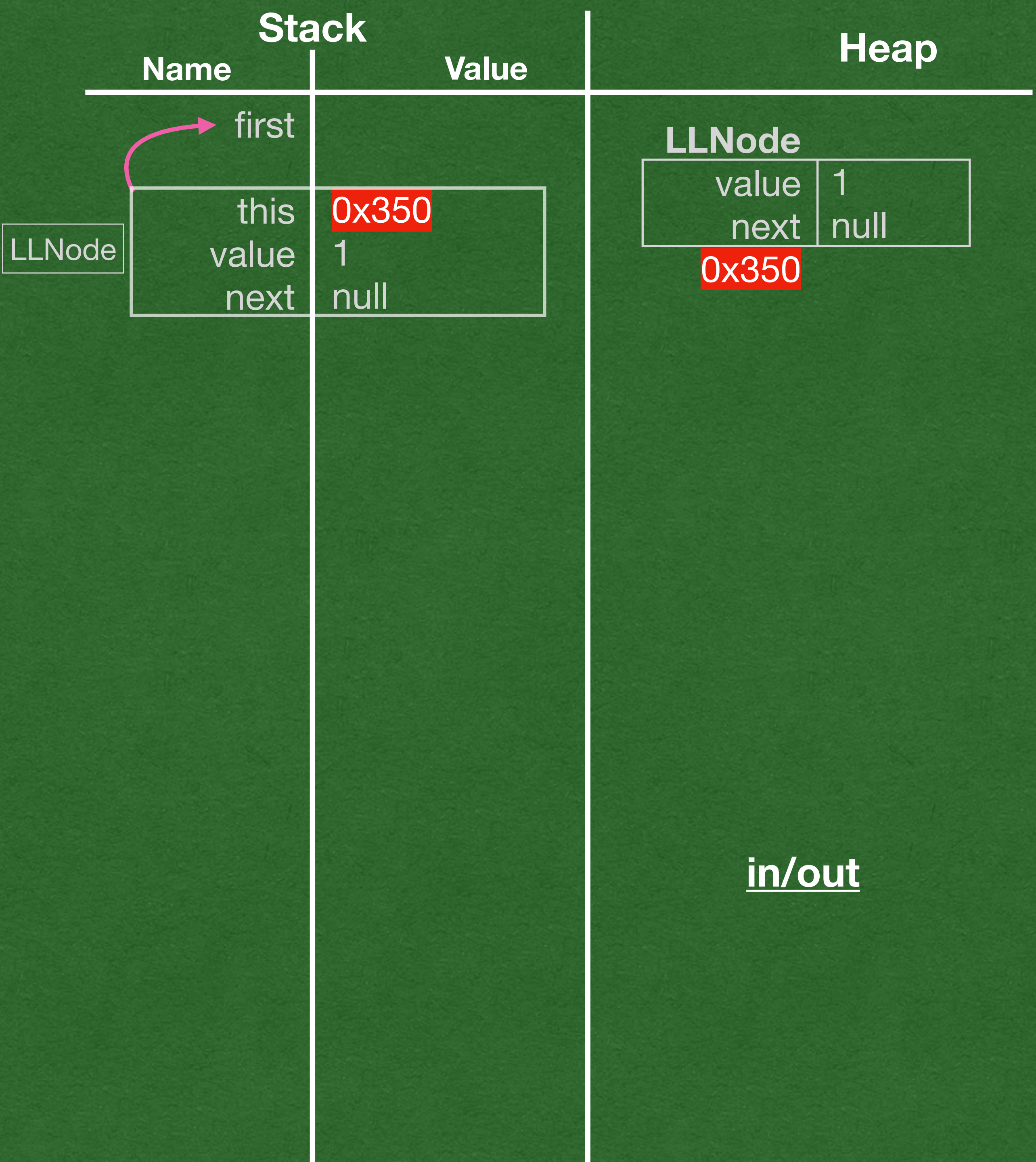
    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    ➡ public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
    }
}
```

[illegible]

- Create a LLNode object
- next is equal to null
 - The lack of a reference

```
public class LLNode {  
    private int value;  
    private LLNode next;  
  
    public LLNode(int value, LLNode next) {  
        this.value = value;  
        this.next = next;  
    }  
  
    public static void main(String[] args) {  
        LLNode first = new LLNode(1, null);  
        first = new LLNode(2, first);  
        first = new LLNode(3, first);  
    }  
}
```

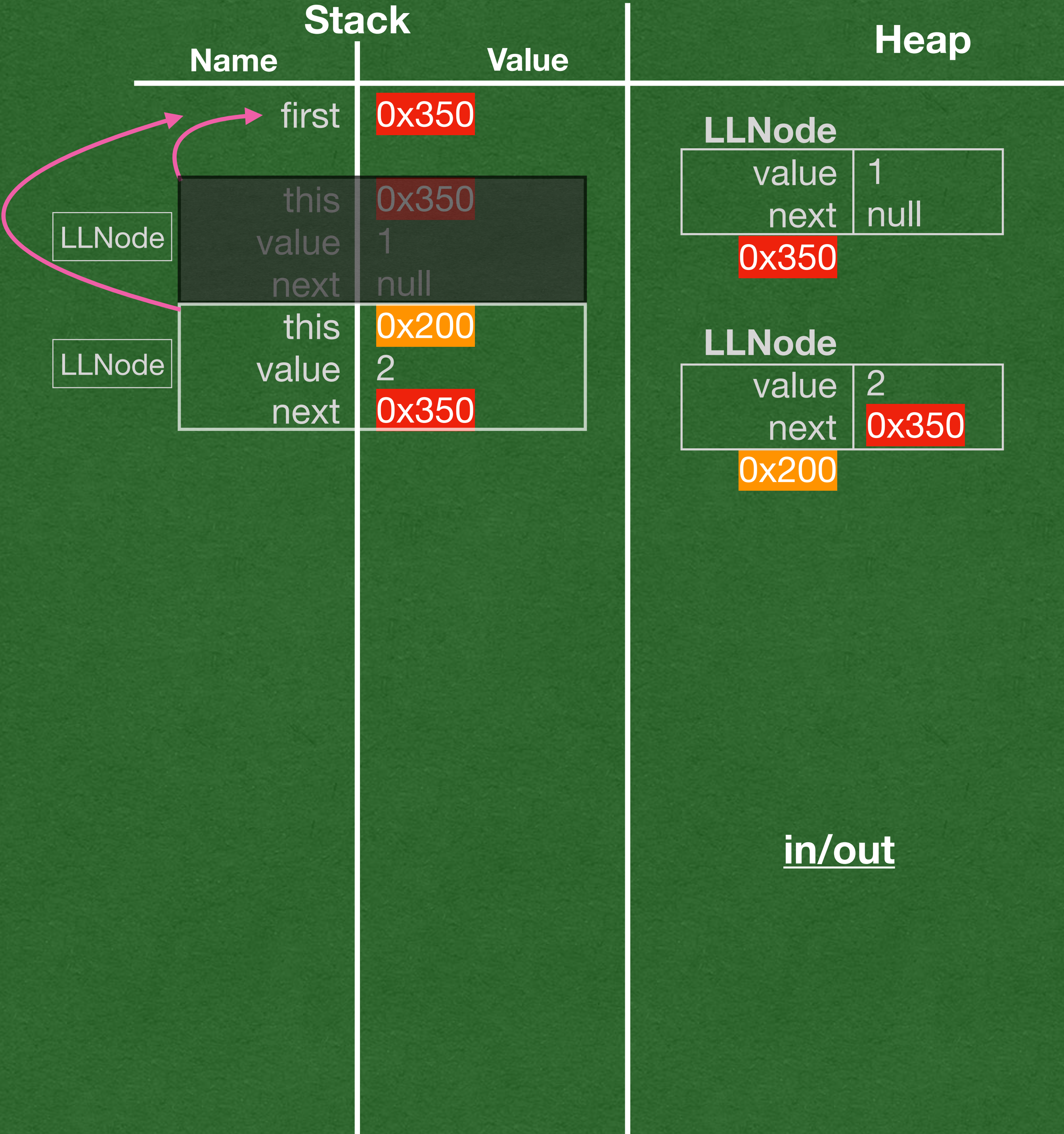


- Call the constructor again
- Pass myList (0x350) as next

```
public class LLNode {
    private int value;
    private LLNode next;

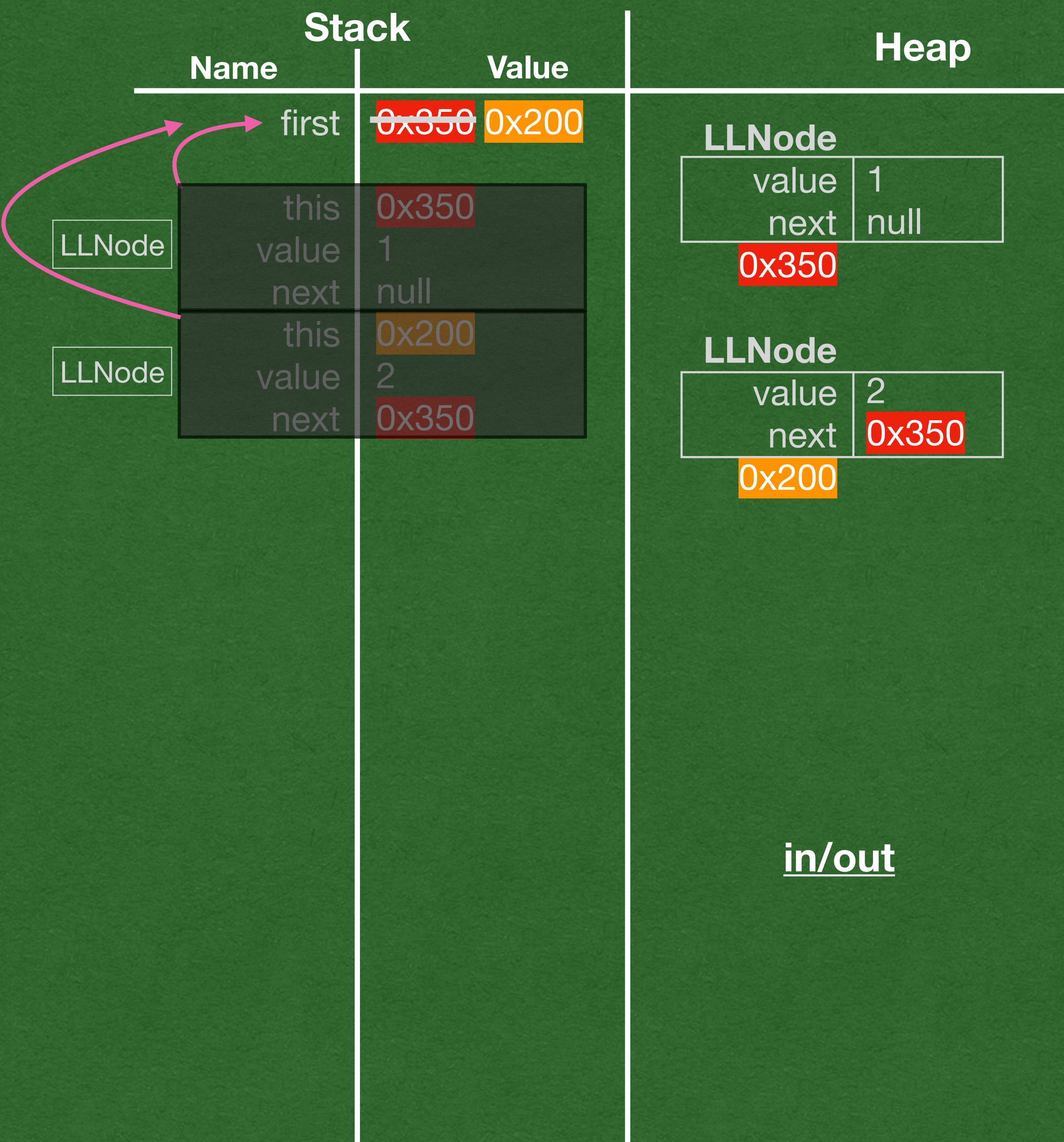
    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
    }
}
```



- Reassign first to the reference returned by the constructor
- first now stores 0x200 which has a next of 0x350

```
public class LLNode {  
    private int value;  
    private LLNode next;  
  
    public LLNode(int value, LLNode next) {  
        this.value = value;  
        this.next = next;  
    }  
  
    public static void main(String[] args) {  
        LLNode first = new LLNode(1, null);  
        first = new LLNode(2, first);  
        first = new LLNode(3, first);  
    }  
}
```



- Repeat the process for the node with value 3
- We now have a linked list with 3 elements

```
public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
    }
}
```



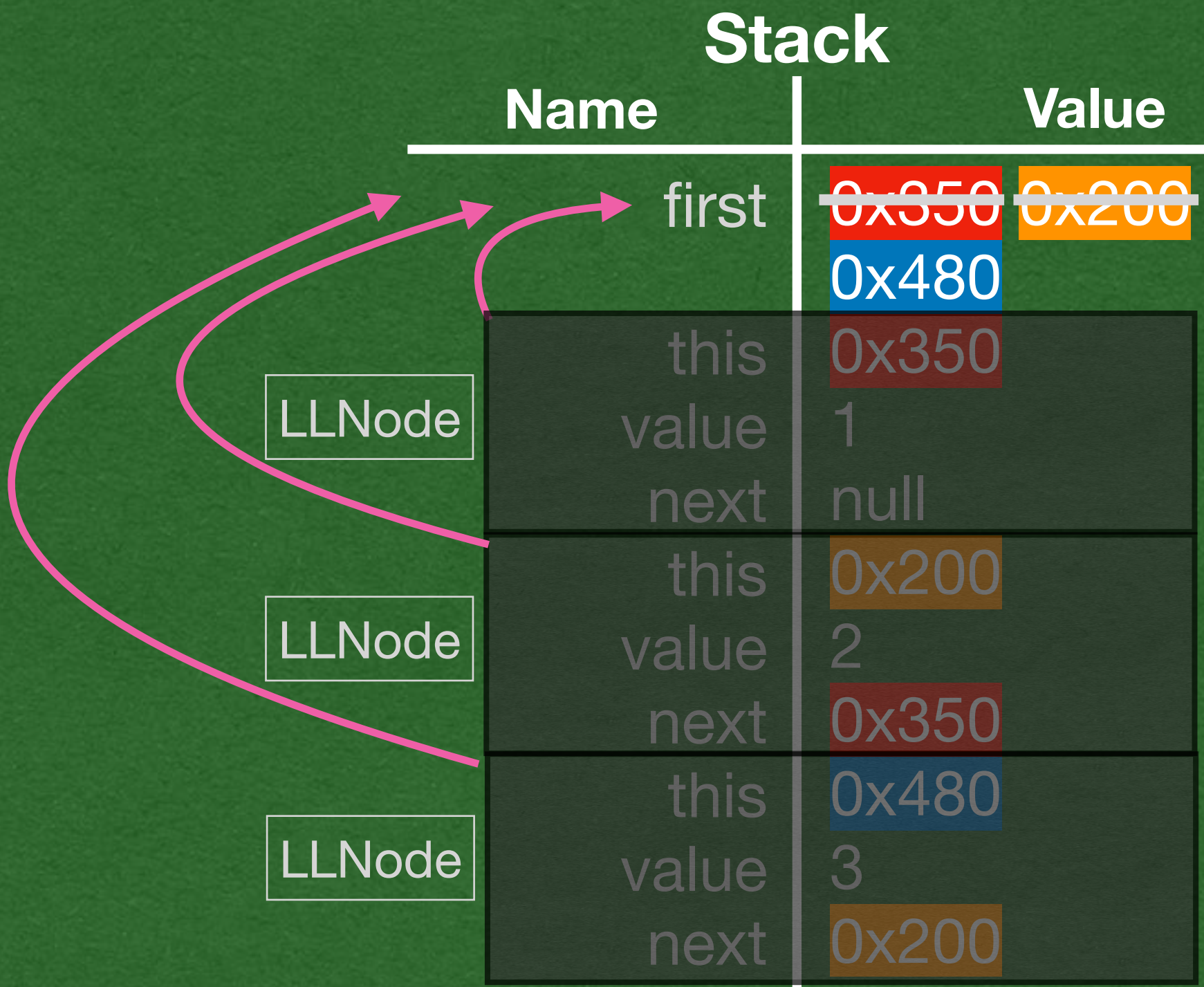
- Our variable "first" only stores a reference to the first node of the list
- We call the first node the head of the list

```
public class LLNode {  
    private int value;  
    private LLNode next;  
  
    public LLNode(int value, LLNode next) {  
        this.value = value;  
        this.next = next;  
    }  
  
    public static void main(String[] args) {  
        LLNode first = new LLNode(1, null);  
        first = new LLNode(2, first);  
        first = new LLNode(3, first);  
    }  
}
```



- Each node stores one value of the list and a reference to the next node
- Each node can be anywhere on the heap

```
public class LLNode {  
    private int value;  
    private LLNode next;  
  
    public LLNode(int value, LLNode next) {  
        this.value = value;  
        this.next = next;  
    }  
  
    public static void main(String[] args) {  
        LLNode first = new LLNode(1, null);  
        first = new LLNode(2, first);  
        first = new LLNode(3, first);  
    }  
}
```



in/out

toString

```
public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int val, LLNode next) {
        this.value = val;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        String value = first.toString();
        System.out.println(value);
    }
}
```

- Let's add a toString method to our Linked List
- This will return the values separated by spaces
- .. aaand it uses recursion!
- Remember recursion?
- We're using it!

2 Memory 2 Diagram

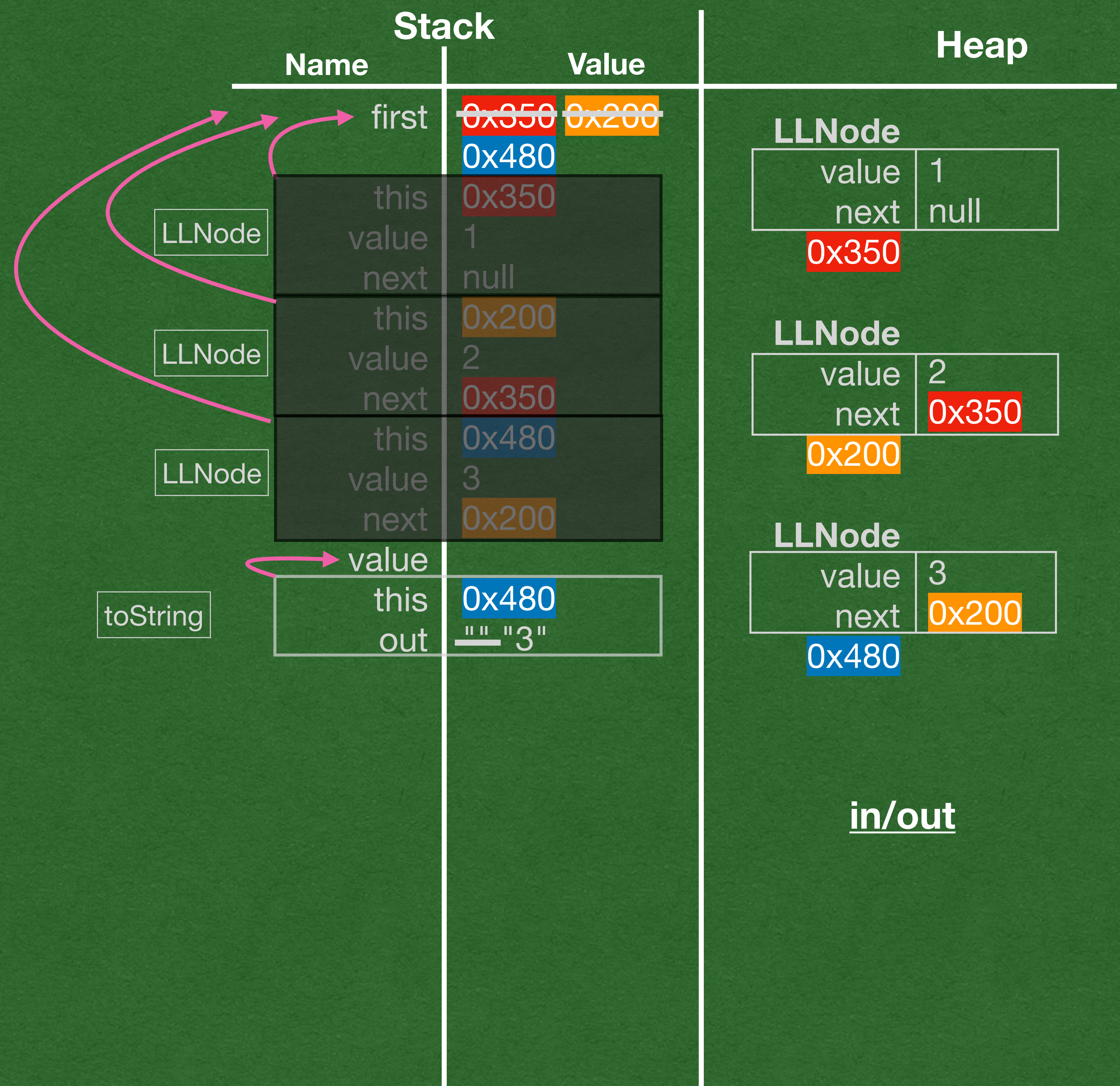
- We could write
 - System.out.println(first)
- We're explicitly calling toString to be clear of our intentions

```
public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        ➔ if (this.next != null) {
            out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        ➔ String value = first.toString();
        System.out.println(value);
    }
}
```



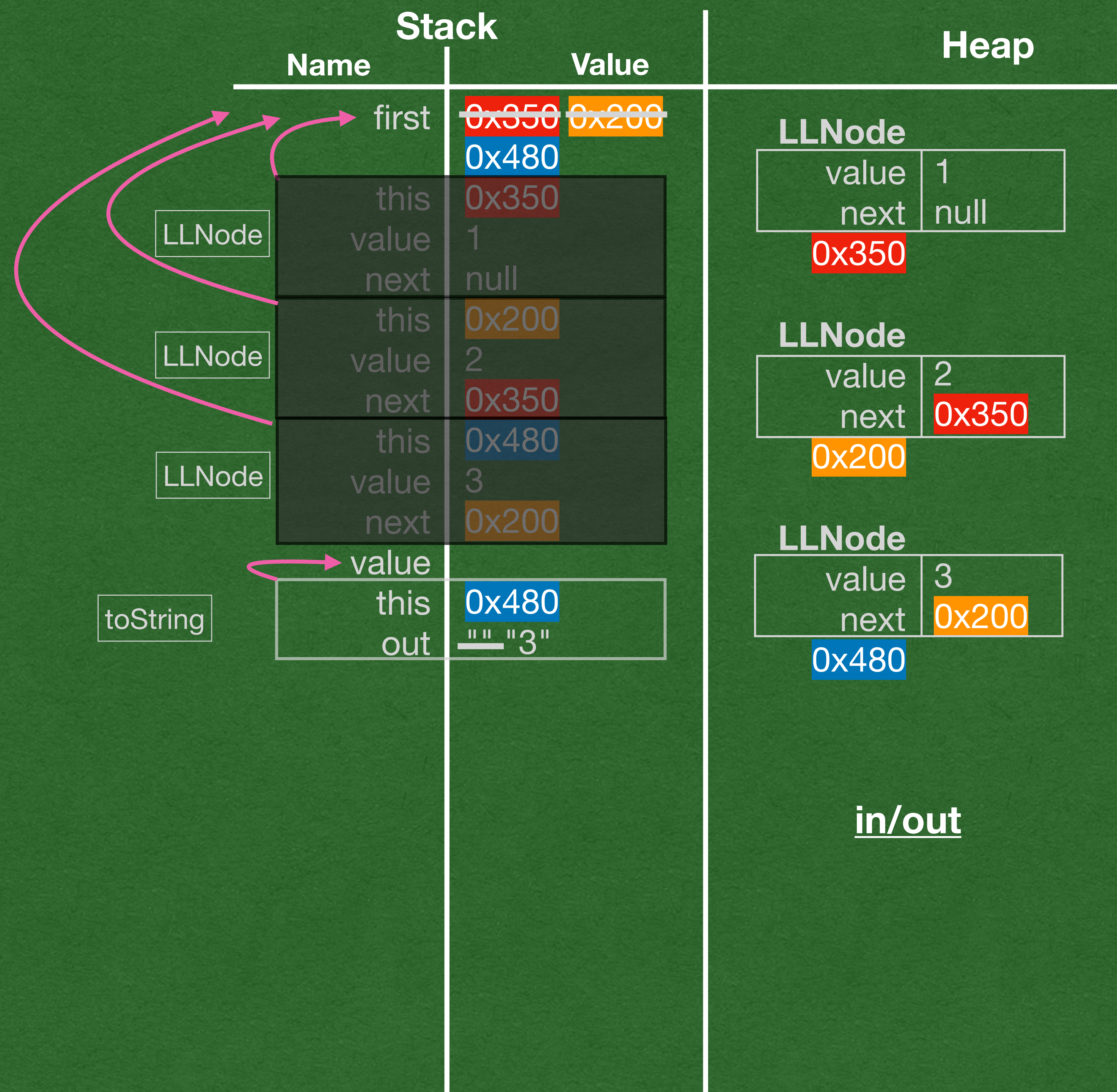
- If next is not null, we are not at the end of the list
- There's more work to be to done
- Make a recursive call

```
public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        String value = first.toString();
        System.out.println(value);
    }
}
```



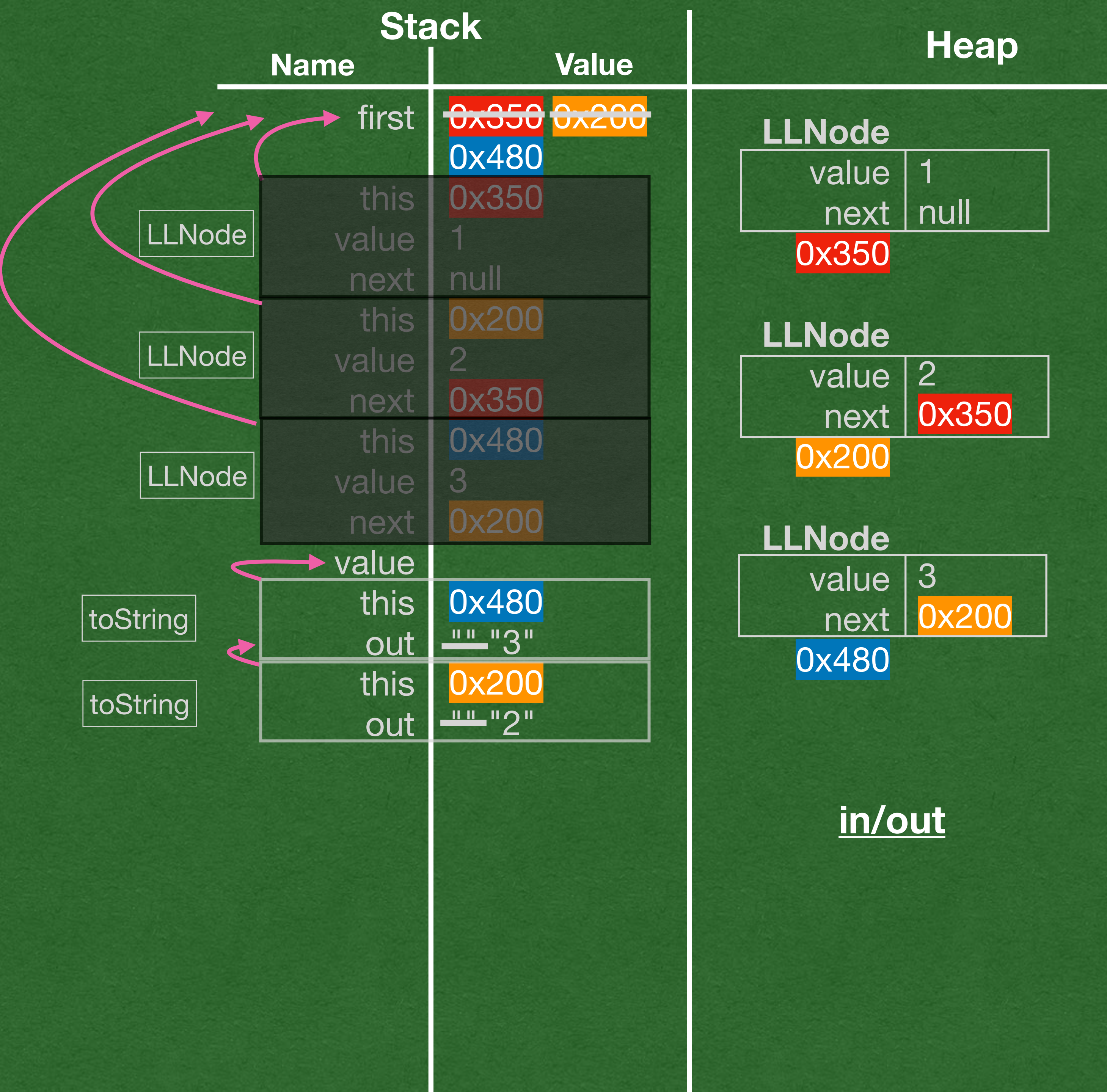
- The recursive call is made on the next node
- The first stack frame waits for the return value of the recursive call

```
public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        String value = first.toString();
        System.out.println(value);
    }
}
```



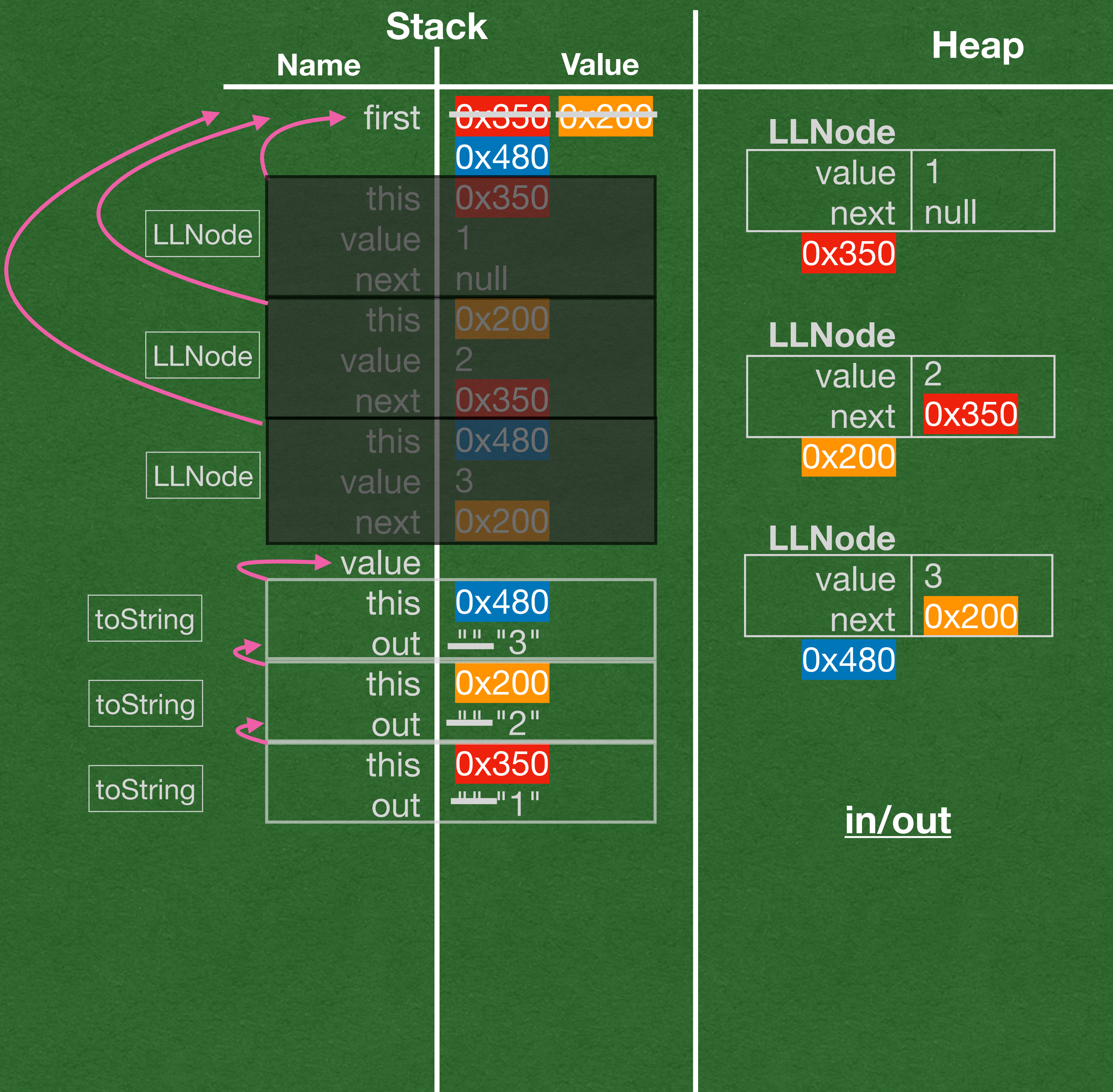
- Make another recursive call
- In this stack frame, the condition is false

```
public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        String value = first.toString();
        System.out.println(value);
    }
}
```



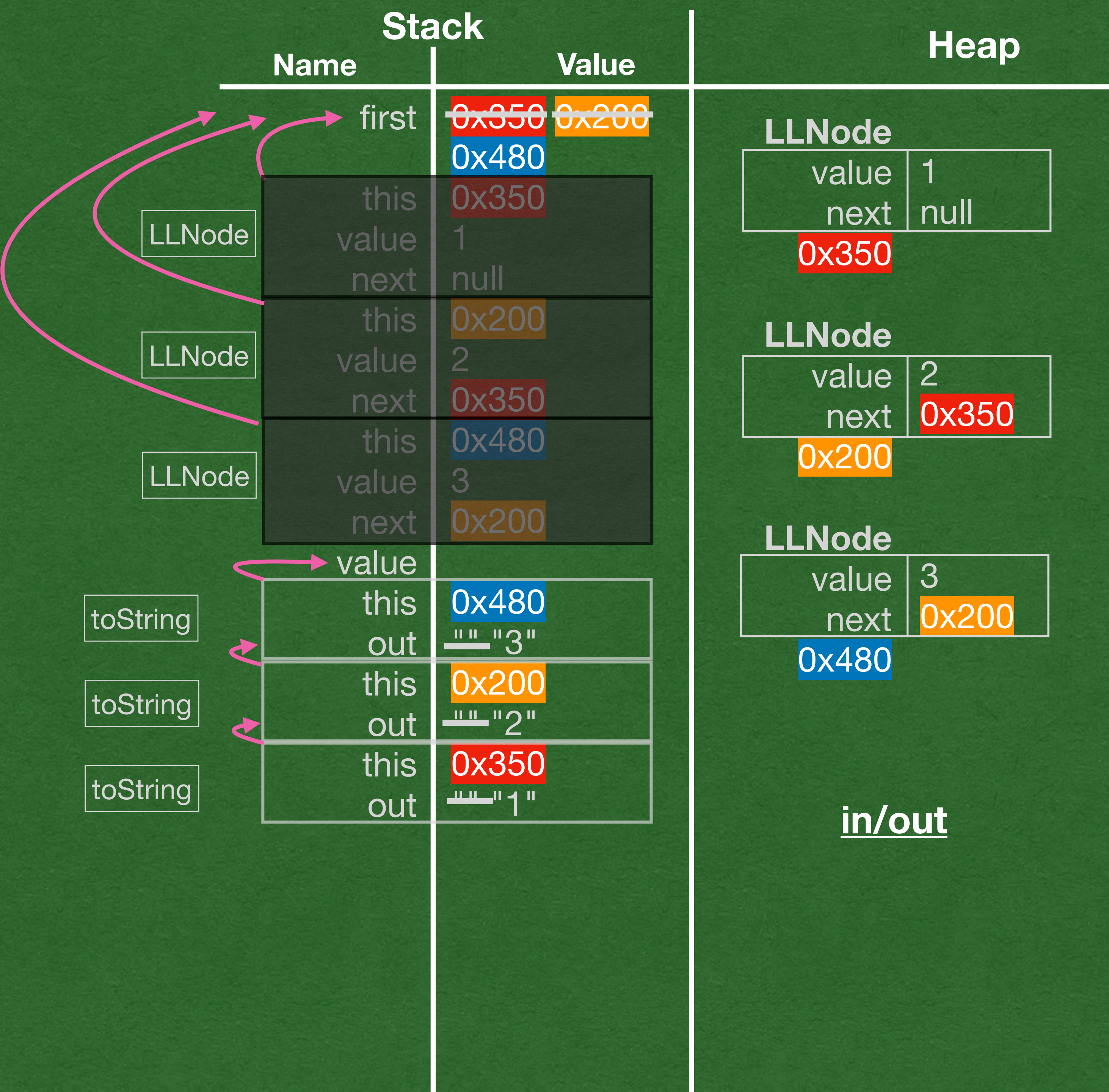
- This frame returns "1" to the previous stack frame

```
public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        String value = first.toString();
        System.out.println(value);
    }
}
```



- The previous stack frame (With this == 0x200) is back on top of the stack
- It takes the return value of "1" and continues running code

```
public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        String value = first.toString();
        System.out.println(value);
    }
}
```



- Return "2 1" to the first recursive stack frame

```
public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        String value = first.toString();
        System.out.println(value);
    }
}
```



- The frame with this == 0x480 is back on top of the stack
- Concatenate the returned value to out

```
public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        String value = first.toString();
        System.out.println(value);
    }
}
```



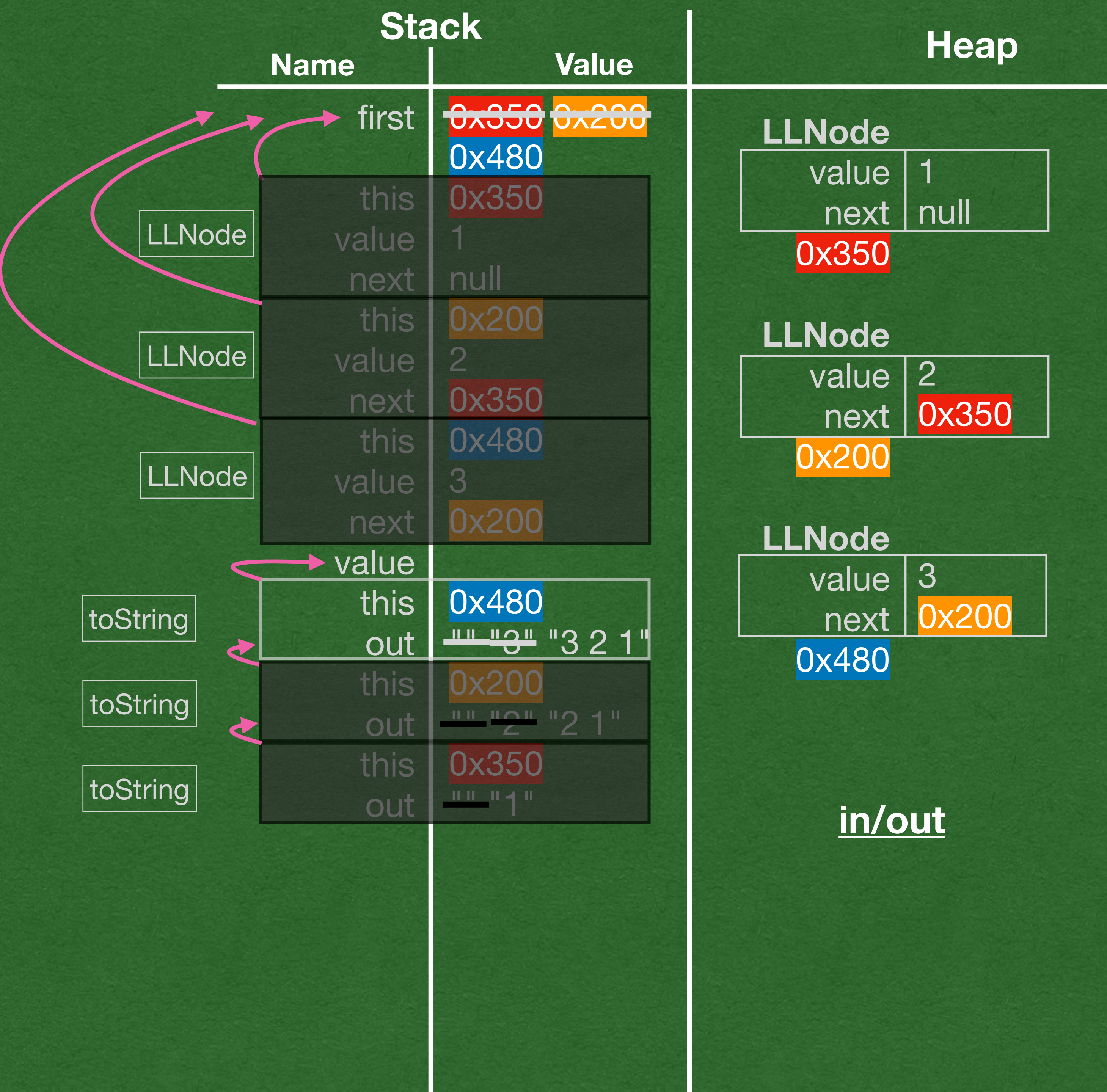
- Return "3 2 1" to the main method

```
public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        String value = first.toString();
        System.out.println(value);
    }
}
```



- Assign "3 2 1" to value in the main stack frame
- We only called toString on the head of the list, but got all the values of the list

```
public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        ➡ String value = first.toString();
        System.out.println(value);
    }
}
```



- Print to the screen and program ends

```
public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        String value = first.toString();
        System.out.println(value);
    }
}
```

